

# Neuronale Netze

## Vorlesung 1, Deep Learning

Dozenten: Prof. Dr. M. O. Franz, Prof. Dr. O. Dürr

HTWG Konstanz, Fakultät für Informatik

# Übersicht

1 Einleitung: Deep Learning

2 Abstrakte Neuronen

3 Künstliche neuronale Netze

4 Tiefe Netze

# Übersicht

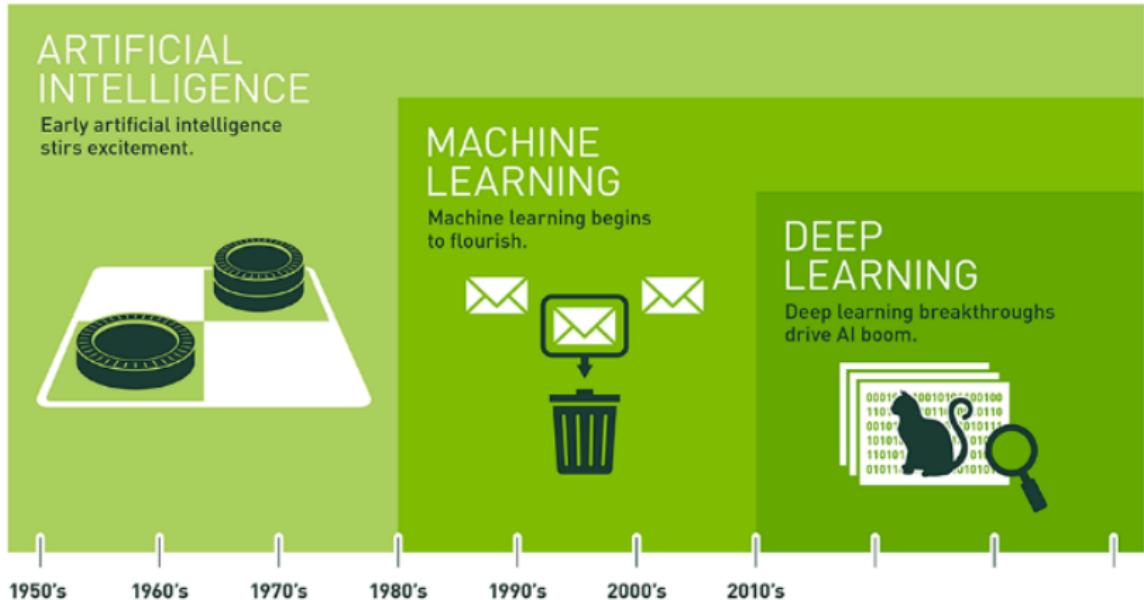
1 Einleitung: Deep Learning

2 Abstrakte Neuronen

3 Künstliche neuronale Netze

4 Tiefe Netze

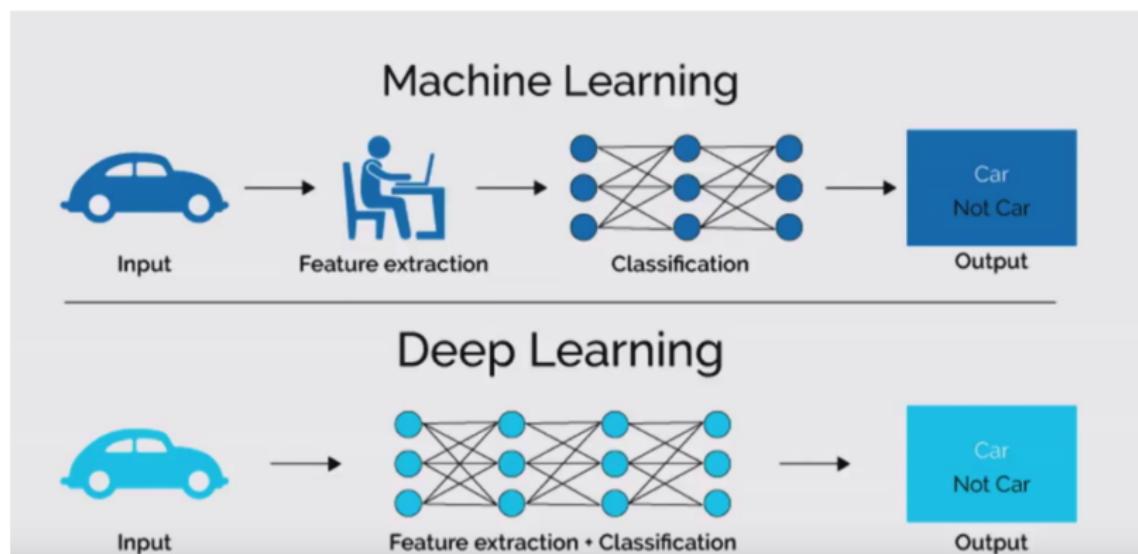
# KI, Maschinelles Lernen und Deep Learning



Slide credit: <https://www.datasciencecentral.com/profiles/blogs/artificial-intelligence-vs-machine-learning-vs-deep-learning>

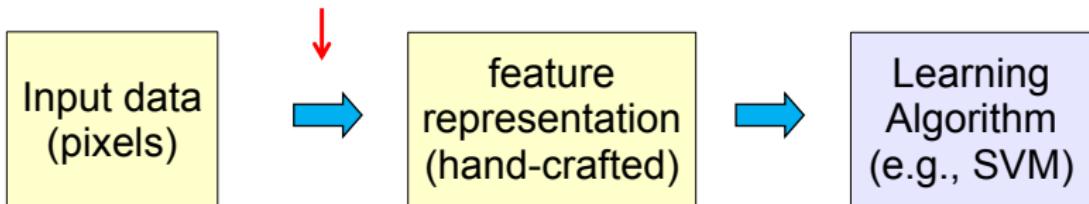
# Deep Learning

**Deep Learning** ist eine spezielle Form des maschinellen Lernens, bei der die Lernmaschine aus multiplen, hintereinander geschalteten Verarbeitungsschichten besteht. Die Schichten extrahieren zunehmend komplexere Merkmale aus dem Input. Diese Merkmale werden gelernt, nicht vorgegeben.



# Traditioneller Ansatz in der Mustererkennung

Features are not learned



Image



Low-level  
vision features  
(edges, SIFT, HOG, etc.)



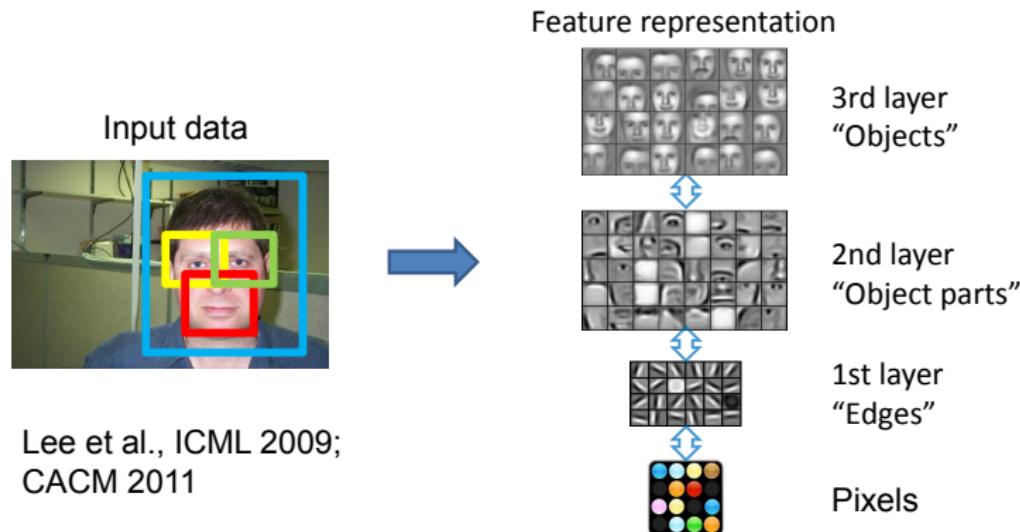
Object detection  
/ classification

[Taylor et al., 2014]

# Grenzen des traditionellen Ansatzes

- Gute Bildmerkmale sind der Schlüssel für eine hohe Erkennungsleistung.
- In den letzten Jahren ist ein ganzer Zoo von *ad hoc* entwickelten Merkmalen entstanden (SIFT, HoG, etc.).
- Die Gewinner von Wettbewerben nutzen oft dutzende Merkmale gleichzeitig, dadurch wird die Rechenzeit immer größer.
- Die verfügbaren traditionellen Klassifikatoren (SVMs, Boosting, Bayesnetze) machen fast keinen Unterschied in der Erkennungsleistung.
- In vielen Einsatzgebieten ist unklar, welche Merkmale man nehmen sollte (z.B. in Texturen, Videos, multispektralen Bildern,...)

# Das Ideal: eine gelernte Merkmalshierarchie



- **Verteilte Repräsentation:** Merkmale in Zwischenschichten werden vielfach wiederverwendet.
- **Grundidee:** eine Schicht extrahiert Merkmale aus der Vorgängerschicht. Leider haben SVMs keine Schichtenstruktur...

# Aufbau der Vorlesung

- Findet in der zweiten Semesterhälfte statt.
- 4 Stunden Vorlesung (ergibt 2 SWS)
- 2 Stunden Praktikum (wöchentlich, ergibt 1 SWS)
- Voraussetzung für die Teilnahme: Teilnahme an der Vorlesung *Machine Learning* in der ersten Semesterhälfte.
- Vorlesung und Praktikum werden mithilfe der Lernplattform Moodle abgehalten. URL:  
<https://moodle.htwg-konstanz.de/moodle/>(Standard HTWG-Rechenzentrum-Login)
- Vorlesungsfolien, ergänzendes Material und Praktikumsunterlagen finden sich unter MSI Informatik, Deep Learning. Bitte manuell einschreiben.

# ECTS-Punkte, Arbeitsaufwand und Prüfung

**Ziel:** Erlernen der Grundlagen, Theorie und Anwendungen von tiefen neuronalen Netzen; Erarbeitung des Verstandnisses für erweiterte Anwendungen wie z.B. AlphaGo.

## **Arbeitsaufwand:**

- 5 ECTS-Punkte
- 45 Stunden Kontaktzeit
- 105 Stunden Vor- und Nachbearbeitung

## **Prüfung:**

- Pünktliche Abgabe aller Praktikumsübungen
- 90-minütige Klausur

# Übersicht

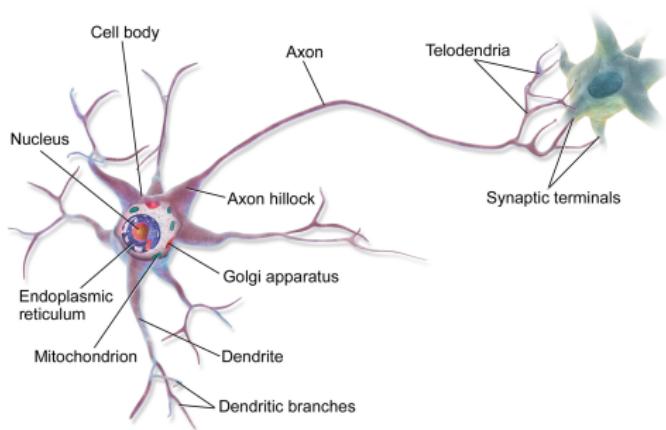
1 Einleitung: Deep Learning

2 Abstrakte Neuronen

3 Künstliche neuronale Netze

4 Tiefe Netze

# Biologische Neuronen



Biologische Neuronen besitzen die drei Grundeigenschaften eines Rechenmodells oder Automaten:

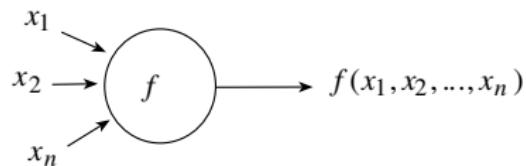
**Informationsübertragung:**  
Information wird entlang der Dendriten, des Zellkörpers und des Axons elektrotonisch oder als Aktionspotential gerichtet übertragen.

[Nvidia]

**Informationsverarbeitung:** Eingehende Signale werden über die Synapsen verstärkt oder abgeschwächt, wechselwirken auf Dendriten und Zellkörper und erzeugen daraus die ausgehenden Signale.

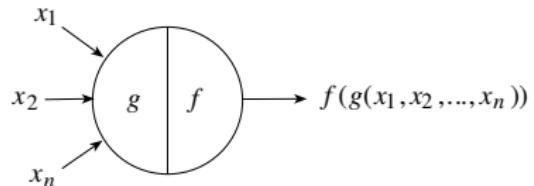
**Informationsspeicherung:** Die Anzahl und Übertragungsstärke der Synapsen kann sich permanent verändern und damit Informationen speichern.

# Abstrakte Neuronen



[Rojas 96]

- Informationsfluss wird über gerichtete Kanten dargestellt.
- Die Veränderung und Verrechnung der Eingangssignale wird als reellwertige Funktion  $f(x_1, x_2, \dots)$  modelliert.



[Rojas 96]

Meist wird die Verrechnung der Signale und die Erzeugung des Ausgangssignals aufgespalten:

- **Integrationsfunktion:**  $g(x_1, x_2, \dots)$  reduziert die  $n$  Eingangssignale zu einem einzigen numerischen Wert  $y$ .
- **Aktivierungsfunktion:**  $f(y)$  berechnet daraus das Ausgangssignal.

Künstliche neuronale Netze (ANNs) sind also gerichtete Graphen mit reellwertigen Funktionen als Knoten.

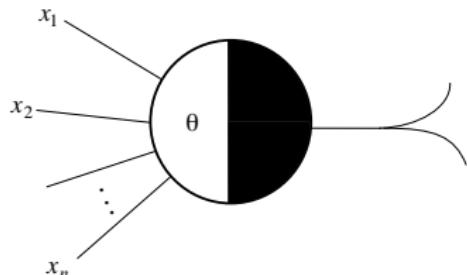
# Was können künstliche neuronale Netze berechnen?

Aufgrund ihrer Eigenschaft als Rechenmodell stellen künstliche neuronale Netze ein biologisches **Berechenbarkeitsparadigma** dar. Weitere bekannte Paradigmen sind:

- Logisch-operationales Modell: Turingmaschinen
- Mathematisches Modell:  $\mu$ -rekursive Funktionen, Lambda-Kalkül
- Computermodell: Registermaschine, Random Access Maschine
- Zelluläre Automaten

Alle genannten Paradigmen sind gleich mächtig, d.h. mit ihnen können alle intuitiv berechenbaren Probleme gelöst werden. Ob das biologische Paradigma gleich mächtig ist, hängt von der Art der Integrations- und Aktivierungsfunktionen ab und von der Topologie des Netzwerks. Tatsächlich reichen schon einfachste abstrakte Neuronen für eine Turing-Vollständigkeit aus.

# Einfachstes abstraktes Neuron: McCulloch-Pitts-Neuron (1943)



[Rojas 96]

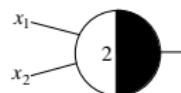
- Verarbeitet und produziert nur binäre Signale.
- Hat exzitatorische und inhibitorische Eingänge.
- Der Schwellwert  $\theta$  gibt an, wieviele exzitatorische Eingänge aktiv sein müssen, damit das Neuron feuert (d.h. Aktivierungsfunktion ist die Sprungfunktion, Integrationsfunktion die Summe).
- Sobald ein inhibitorischer Eingang aktiv ist, wird jeglicher Output unterdrückt.

Es gibt keine Gewichte, nur Verbindungen. Die Kombination von McCulloch-Pitts-Neuronen untersucht die **Schwellwertlogik**.

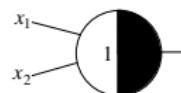
# Implementierung logischer Funktionen

Mit exzitatorischen Verbindungen lassen wichtige, aber nicht alle logischen Funktionen nachbilden:

AND



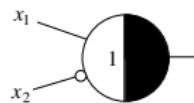
OR



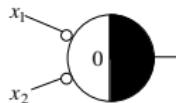
[Rojas 96]

Lässt man auch inhibitorische Verbindungen zu, erhält man auch NAND-Gatter und damit alle anderen logischen Funktionen:

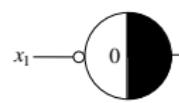
$x_1$  AND  $\neg x_2$



NOR



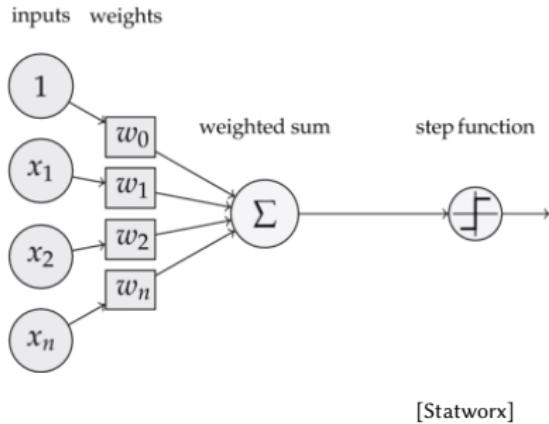
NOT



[Rojas 96]

Leider gibt es keinen effizienten Trainingsalgorithmus.

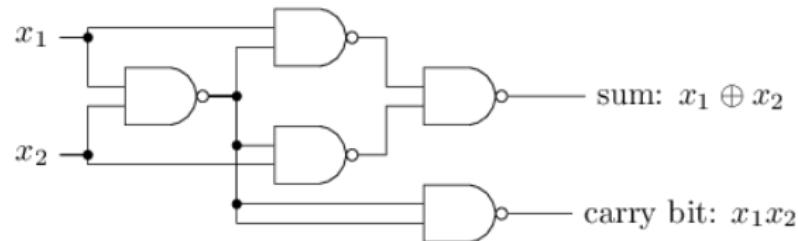
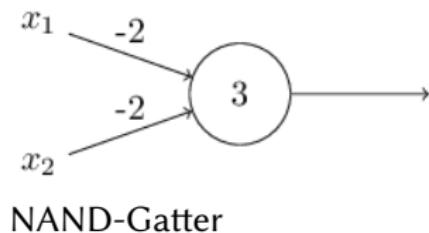
# Ein alter Bekannter: das Perzeptron (1958)



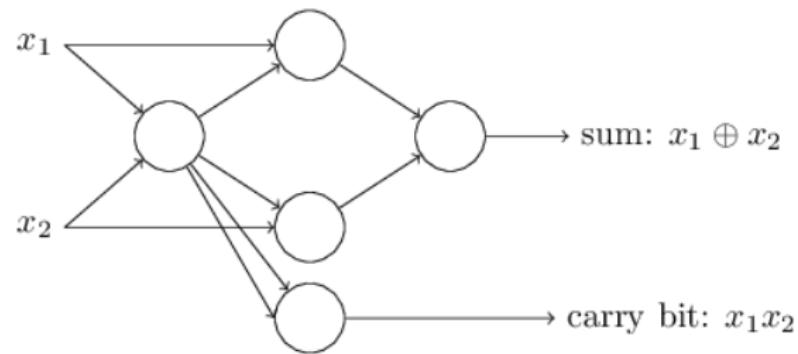
- Erweitert die McCulloch-Pitts-Neuronen auf reelle Ein- und Ausgangssignale und gewichtete Eingangssignale.
- Integrationsfunktion: gewichtete Summe; Aktivierungsfunktion: reeller Schwellwert.
- McCulloch-Pitts-Neuronen lassen sich als Sonderfall eines Perzeptrons darstellen. Somit können auch Perzeptron-Netzwerke jede mögliche logische Funktion implementieren.

Wie bereits in der Vorgängervorlesung gezeigt, entspricht das Perzeptron einem linearen Klassifikator. Die reellen Eingangs- und Ausgangswerte repräsentieren durchschnittliche Feuerraten.

# Beispiel: Perzeptron als NAND-Gate



Addierer aus NAND-Gattern



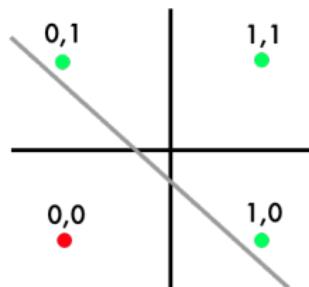
[Nielsen 19]

Addierer aus Perzeptronen

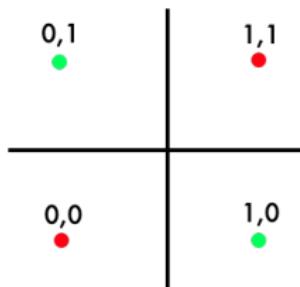
# Das XOR-Problem

Ein einzelnes Perzeptron kann nicht alle logischen Funktionen darstellen. Wie wir bereits aus der Vorgängervorlesung wissen, müssen diese dazu **linear trennbar** sein. Lösbar ist dieses Problem nur mit mehreren gekoppelten Perzeptronen.

The XOR problem



OR



XOR

# Übersicht

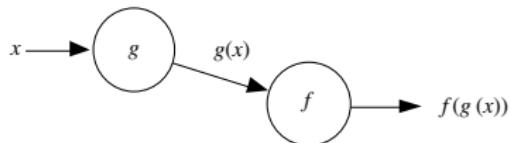
1 Einleitung: Deep Learning

2 Abstrakte Neuronen

3 Künstliche neuronale Netze

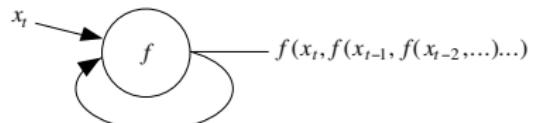
4 Tiefe Netze

# Netzwerktopologie: Feedforward vs. rekurrent



[Rojas 96]

- Bei einer **Feedforward-Architektur** hat der dem Netzwerk zugrundeliegende Graph keine Zyklen.
- Die Modellierung erfolgt durch Verkettung von Funktionen.
- Alle Berechnung erfolgen ohne zeitliche Verzögerungen.



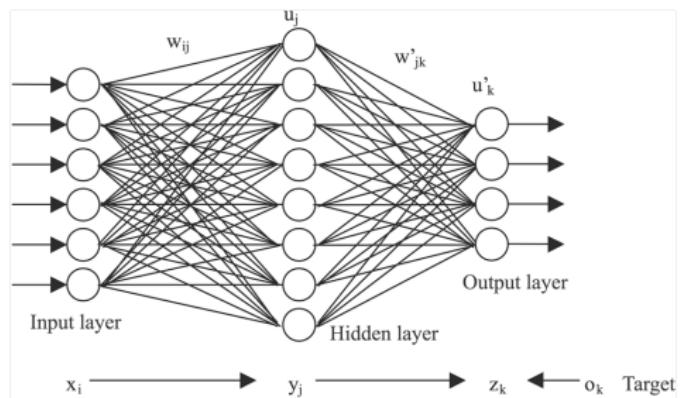
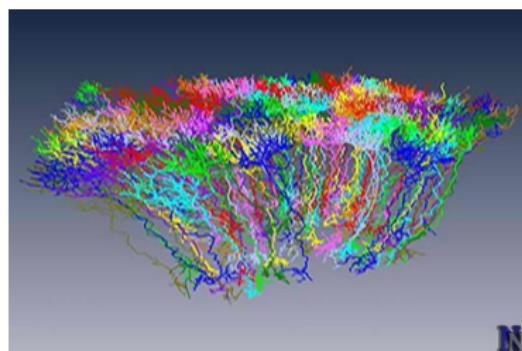
[Rojas 96]

- Rekurrente oder rückgekoppelte Netze haben Zyklen.
- Die Berechnung ist durch die Konnektivität nicht eindeutig bestimmt.
- Man muss zusätzlich das Zeitverhalten der Neuronen spezifizieren.

In der ersten Vorlesungshälfte beschränken wir uns auf Feedforward-Netze.

# Multilayer-Perzeptron

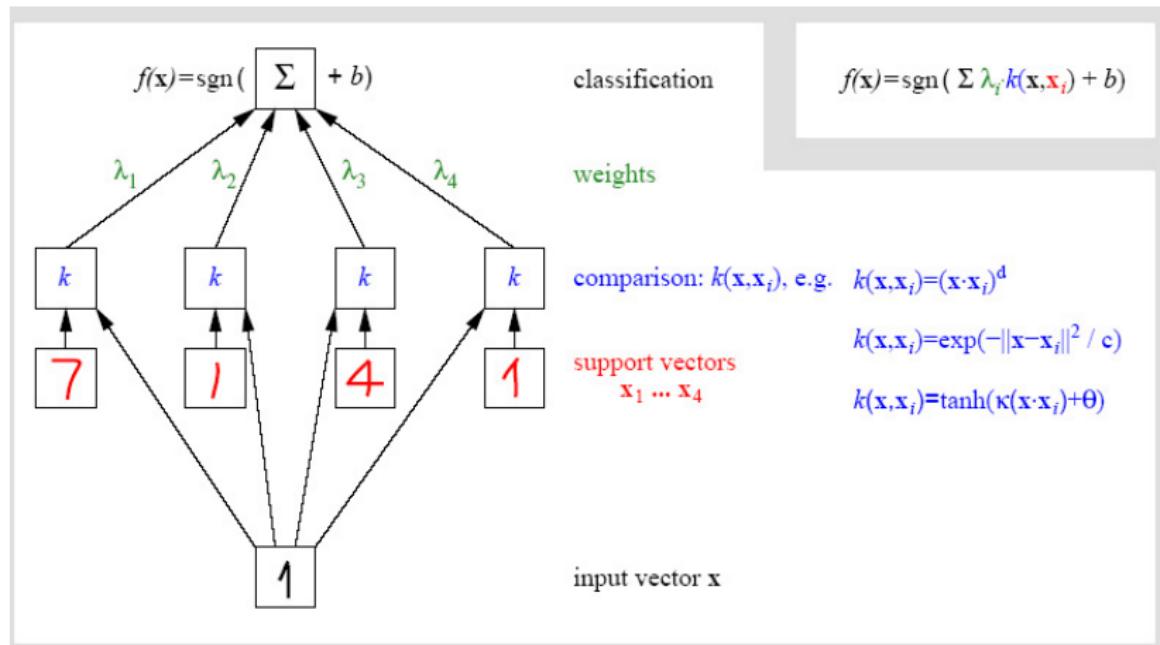
Wie häufig im Gehirn sind die abstrakten Neuronen meist in Schichten angeordnet. Eine solche Architektur wird als *Multilayer-Perzeptron (MLP)* bezeichnet.



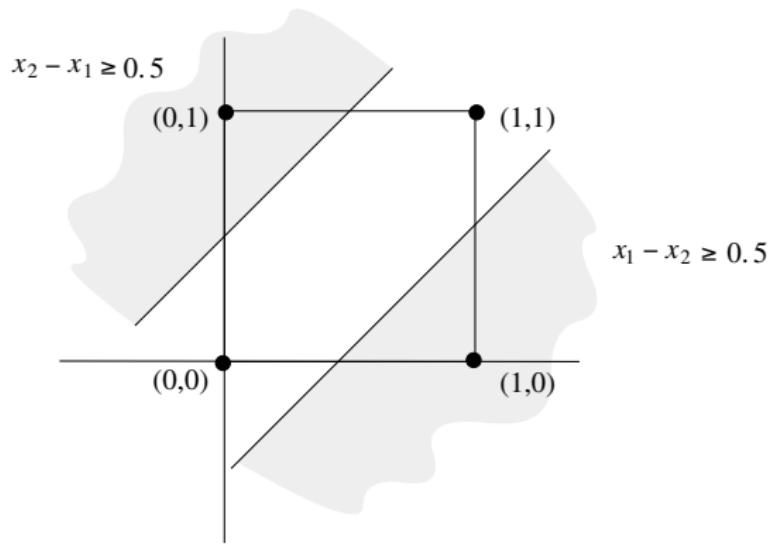
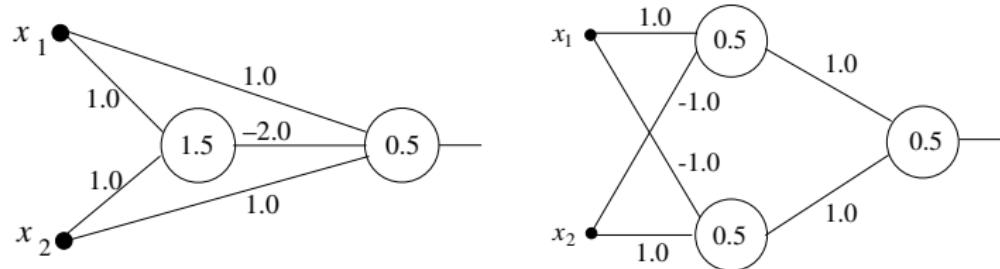
[ExtremeTech.org]

Jedes Netz hat eine Eingangs- und eine Ausgangsschicht und eine oder mehrere versteckte Schichten (*hidden layers*).

# Beispiel: SVM als dreischichtiges neuronales Netz



# XOR revisited



[Rojas 96]

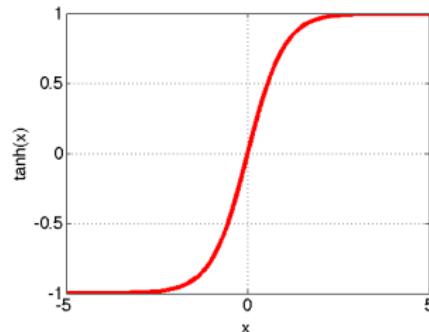
# Sigmoide Neuronen

Sprungfunktionen als Aktivierungsfunktion führen bei neuronalen Netzen zu instabilem Verhalten: eine kleine Änderung des Inputs kann zum kompletten Umkippen des Netzwerkoutputs führen. Wenn das Netzwerk *lernen* soll, dann sollten kleine Änderungen am Eingang auch zu kleinen Änderungen am Ausgang führen.

Erreicht wird dies durch eine differenzierbare Aktivierungsfunktion, z.B.

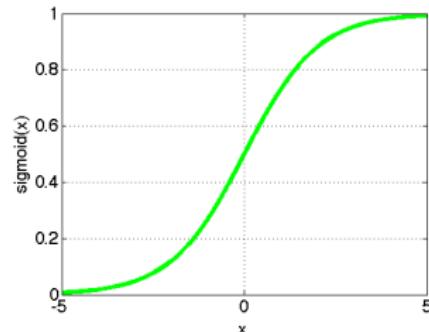
**Sigmoid:**

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

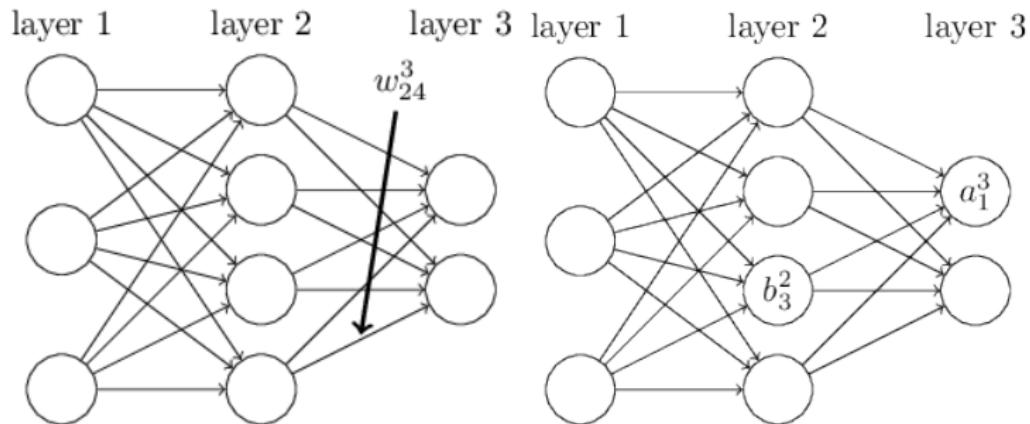


**Tangens Hyperbolicus:**

$$\sigma(z) = \tanh(z)$$



# Neuronale Netze: Notation



[Nielsen, 2016]

$w_{jk}^l$  ist das Gewicht vom  $k$ -ten Neuron der Eingangsschicht  $l - 1$  zum  $j$ -ten Neuron der Schicht  $l$ ,  $b_j^l$  sein Schwellwert,  $a_j^l$  sein Ausgangswert:

$$a_j^l = \sigma \left( \sum_k w_{jk}^l a_k^{l-1} + b_j^l \right).$$

## Matrixnotation

Die Aktivierungen  $a_j^l$  der Schicht  $l$  werden in einem **Aktivierungsvektor**  $a^l$  zusammengefasst, genauso die Schwellwerte  $a_j^l$  in einem **Schwellwertvektor**  $b^l$ . Die Gewichte für das  $j$ -te Neuron bilden die  $j$ -te Reihe der **Gewichtsmatrix**  $w^l$ .

Unter der Annahme, dass  $\sigma(z)$  *vektoriert* ist (d.h. auf jede Komponente des Eingangsvektors angewandt wird), ergibt sich die vereinfachte Matrixschreibweise:

$$a^l = \sigma(w^l a^{l-1} + b^l).$$

Die Zwischengröße  $z^l = w^l a^{l-1} + b^l$  wird oft gebraucht und bezeichnet den **gewichteten Input**.

Alle Operationen lassen sich sehr effektiv in Python, C++, Matlab... implementieren.

# Übersicht

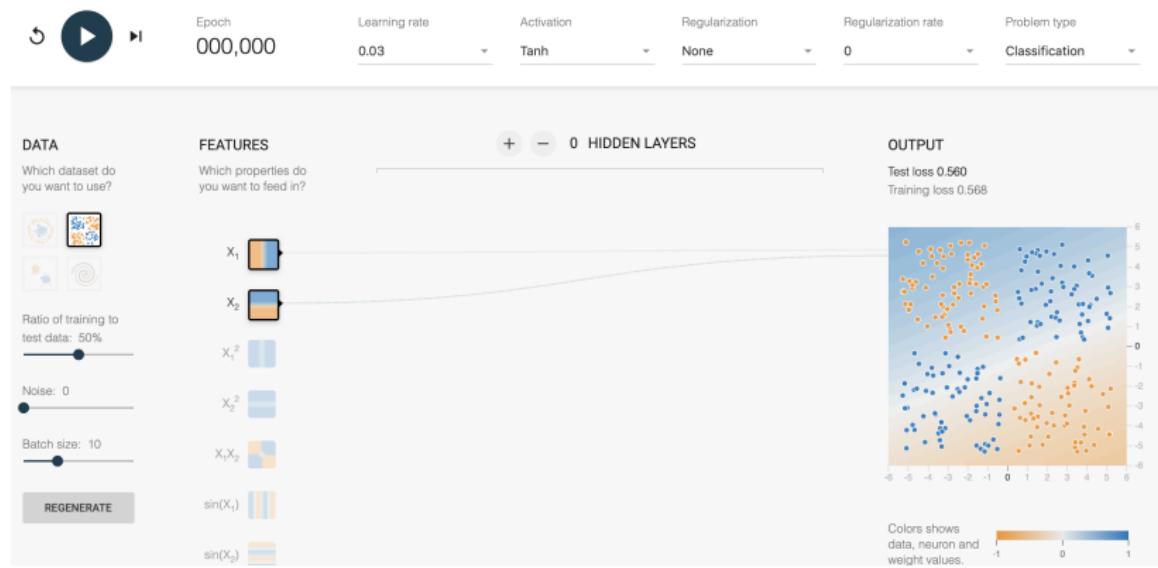
1 Einleitung: Deep Learning

2 Abstrakte Neuronen

3 Künstliche neuronale Netze

4 Tiefe Netze

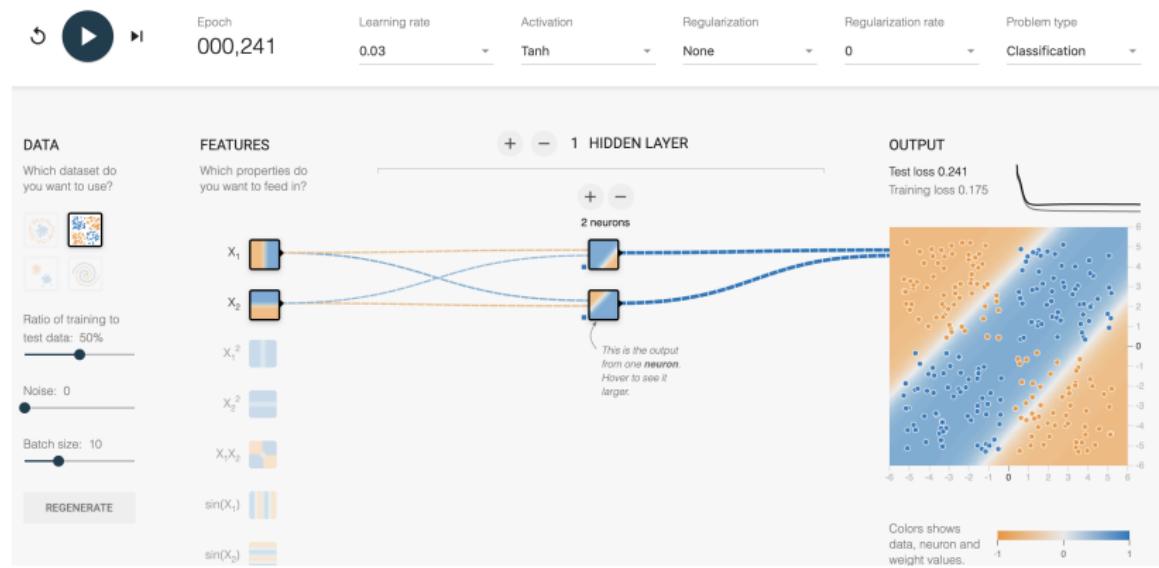
# Experiment: ohne versteckte Schichten (Perzeptron)



[playground.tensorflow.org]

Wir erhalten einen klassischen linearen Klassifikator, aber das Lernproblem ist nicht linear trennbar.

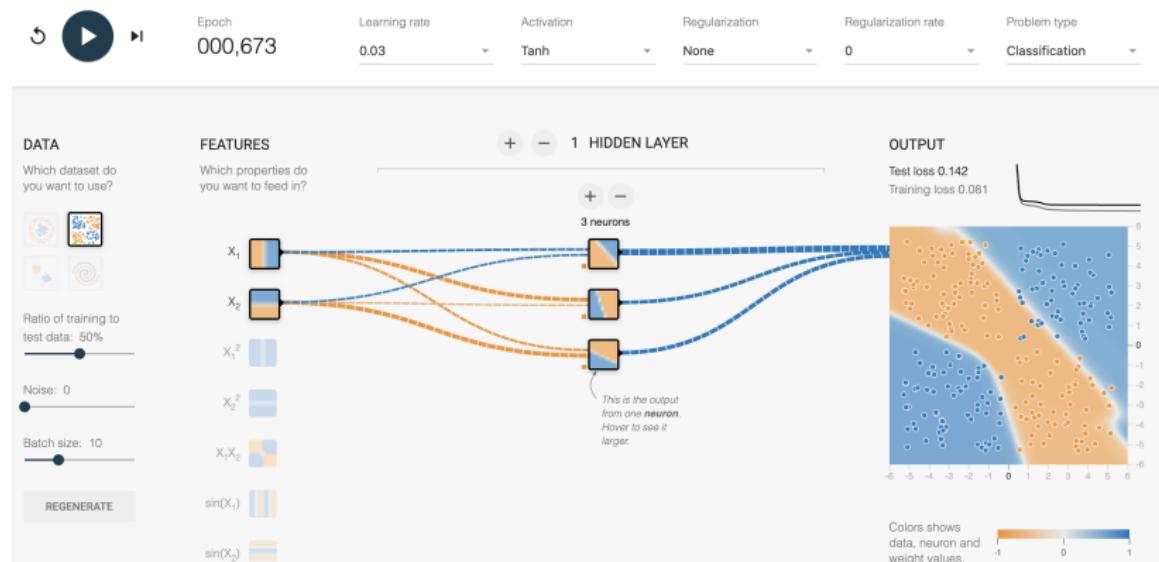
# Experiment: eine versteckte Schicht mit 2 Neuronen



[playground.tensorflow.org]

Entspricht MLP aus dem XOR-Beispiel; trennt die Daten besser, aber kann sich noch nicht an die Quadranten anpassen.

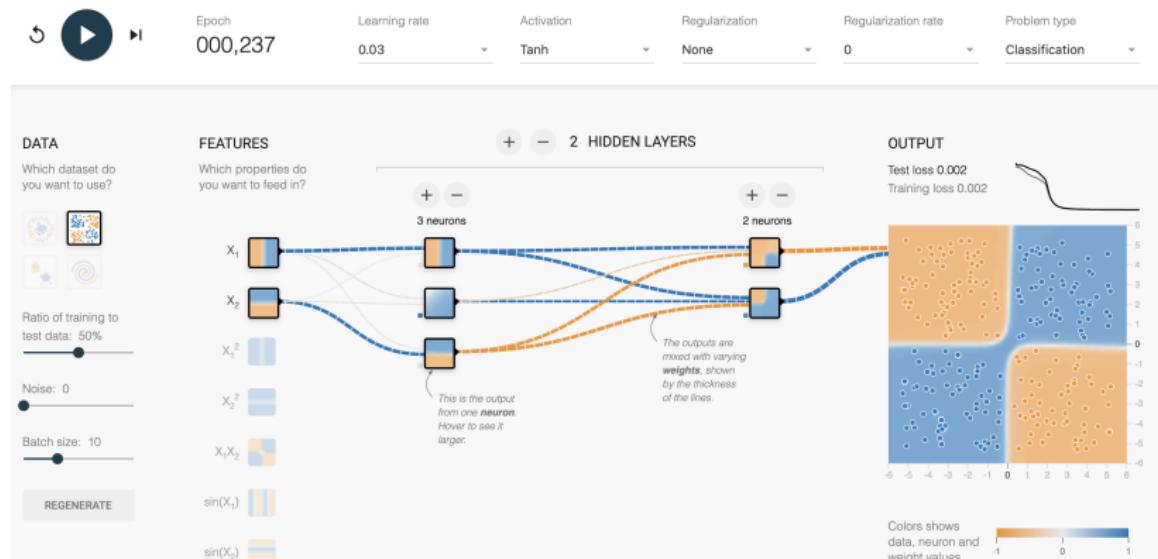
# Experiment: eine versteckte Schicht mit 3 Neuronen



[playground.tensorflow.org]

3 versteckte Neuronen ermöglichen komplexere Entscheidungsgrenzen.

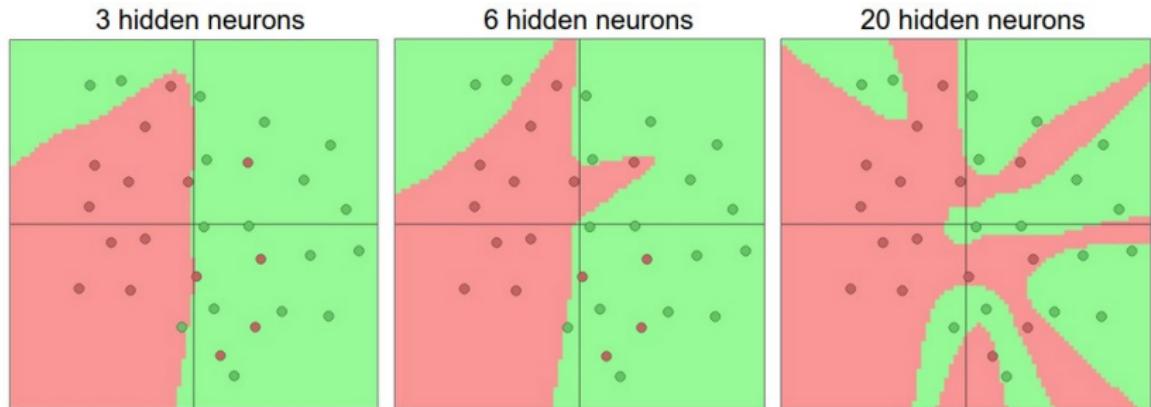
# Experiment: zwei versteckte Schichten



Eine weitere versteckte Schicht führt zu einer perfekten Anpassung.

Hausaufgabe: vollziehen Sie das Experiment nach unter  
<http://playground.tensorflow.org>.

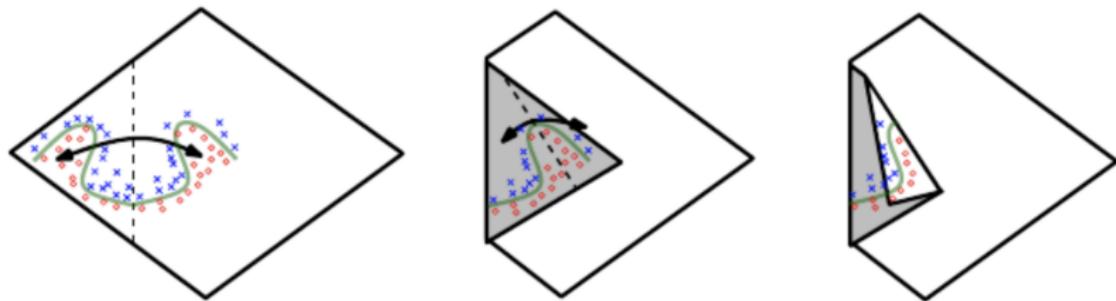
# Approximationsfähigkeit eines MLP mit einer versteckten Schicht



[<http://cs231n.github.io/neural-networks-1/>]

Man kann zeigen, dass eine versteckte Schicht bereits ausreicht, um beliebige Funktionen zu approximieren, solange man die Zahl der Neuronen beliebig erhöhen darf, d.h. ein solches Netz ist ein **universaler Funktionsapproximator**. Warum sollte man dann die Anzahl der versteckten Schichten weiter erhöhen?

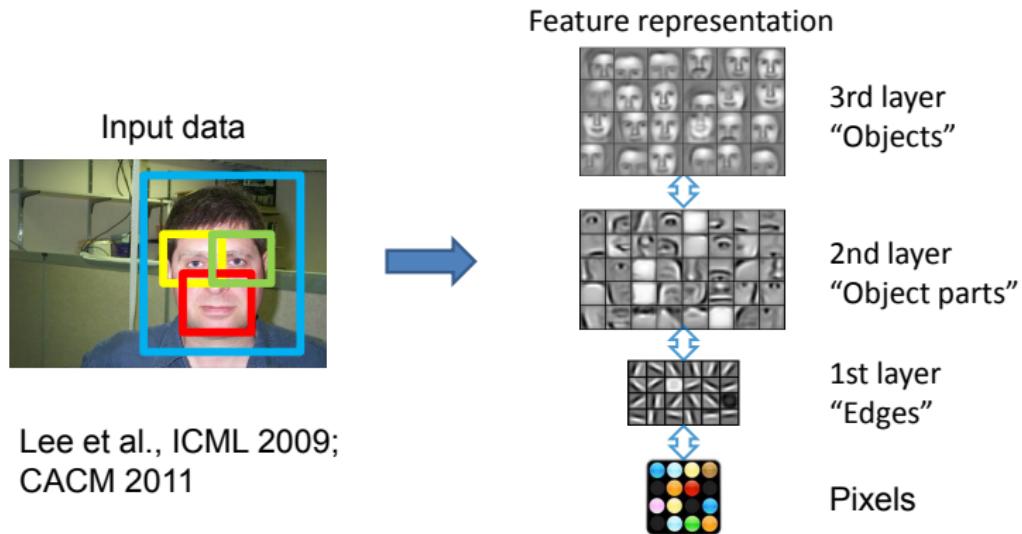
# Tiefe Netze können hierarchische Strukturen effektiv darstellen



[Montufar 2014]

Hierarchisch hintereinander geschaltete Neuronen, die großräumige Symmetrien erkennen können, vereinfachen das Lernproblem erheblich. Will man solche Strukturen mit flachen Netzwerken darstellen, braucht man exponentiell viele Neuronen in der versteckten Schicht.

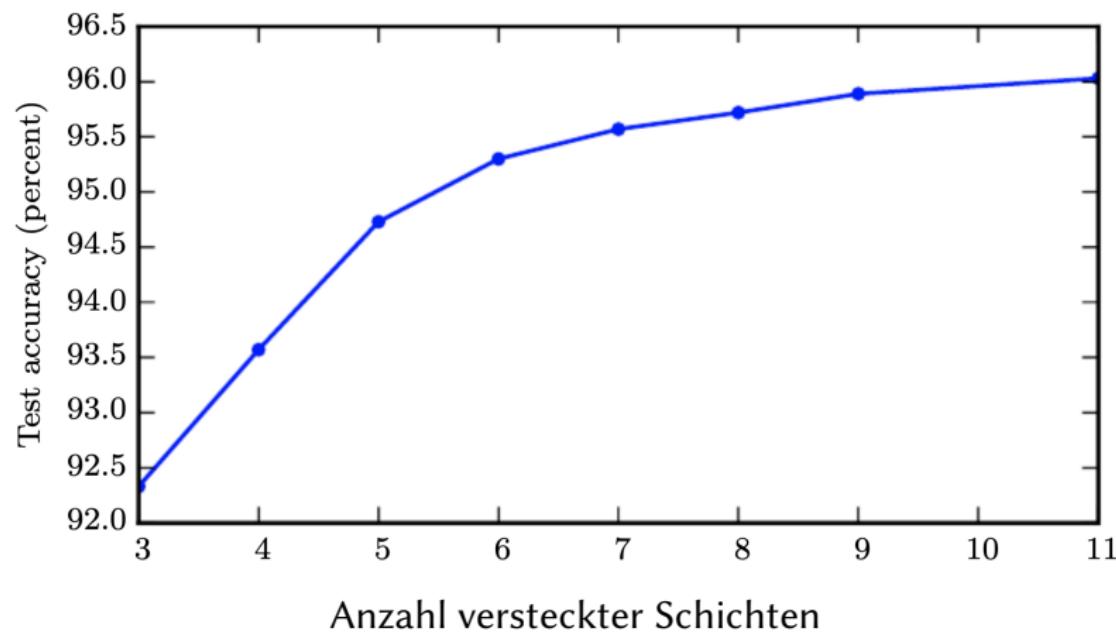
# Tiefe Netze als statistische A-priori-Annahme



Lee et al., ICML 2009;  
CACM 2011

Die Auswahl einer Lernmaschine wird von Annahmen über die Problemstruktur bestimmt. Ein tiefes Netzwerk impliziert eine Problemstruktur, bei der sich die zu lernende Funktion aus einfacheren Funktionen zusammensetzt, diese wiederum aus noch einfacheren usw. Viele praktische Probleme haben diese Struktur.

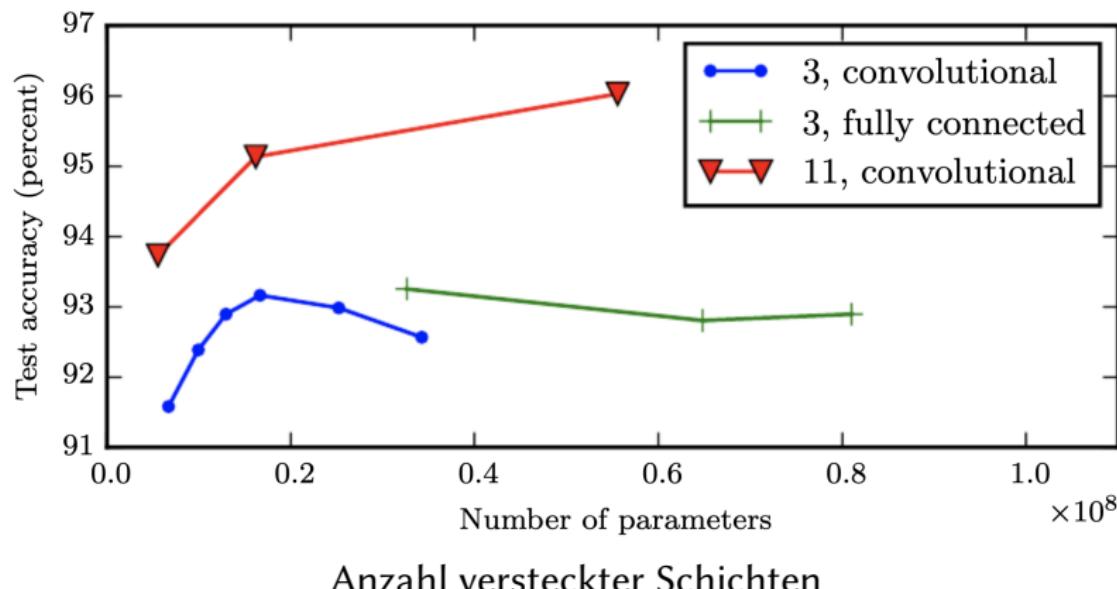
# Empirische Hinweise am Beispiel Ziffernerkennung (1)



[ Goodfellow et al. 2014]

Oft führt eine Erhöhung der Anzahl der Schichten zu einer verbesserten Fehlerrate.

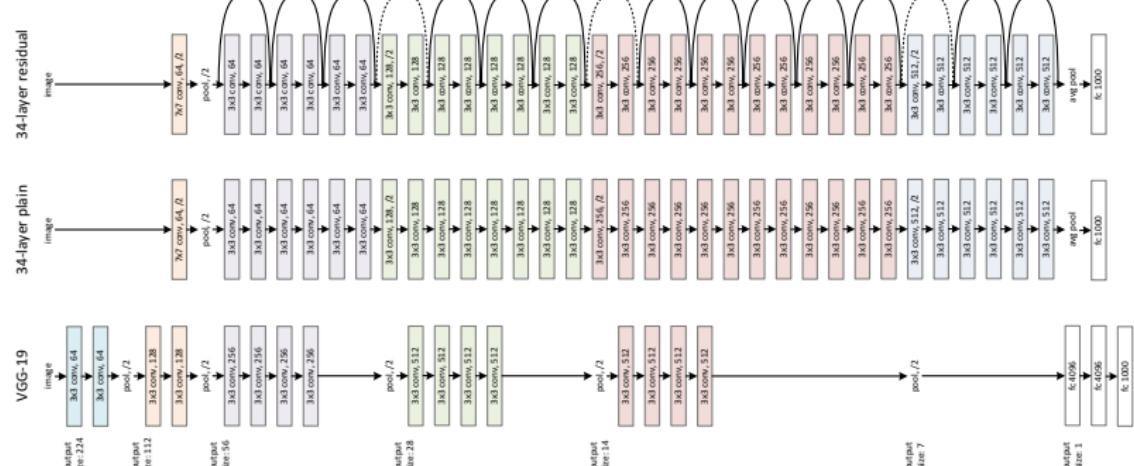
## Empirische Hinweise am Beispiel Ziffernerkennung (2)



[ Goodfellow et al. 2014]

Die verbesserte Leistung geht dabei nicht auf eine Erhöhung der Anzahl der Gewichte zurück.

# Extrembeispiel Resnet



[ <https://arxiv.org/abs/1512.03385> ]

## Training tiefer Netze



memegene

Bis 2006 war das Training tiefer Netze (bis auf wenige Ausnahmen) unmöglich ⇒ nächste Vorlesung...